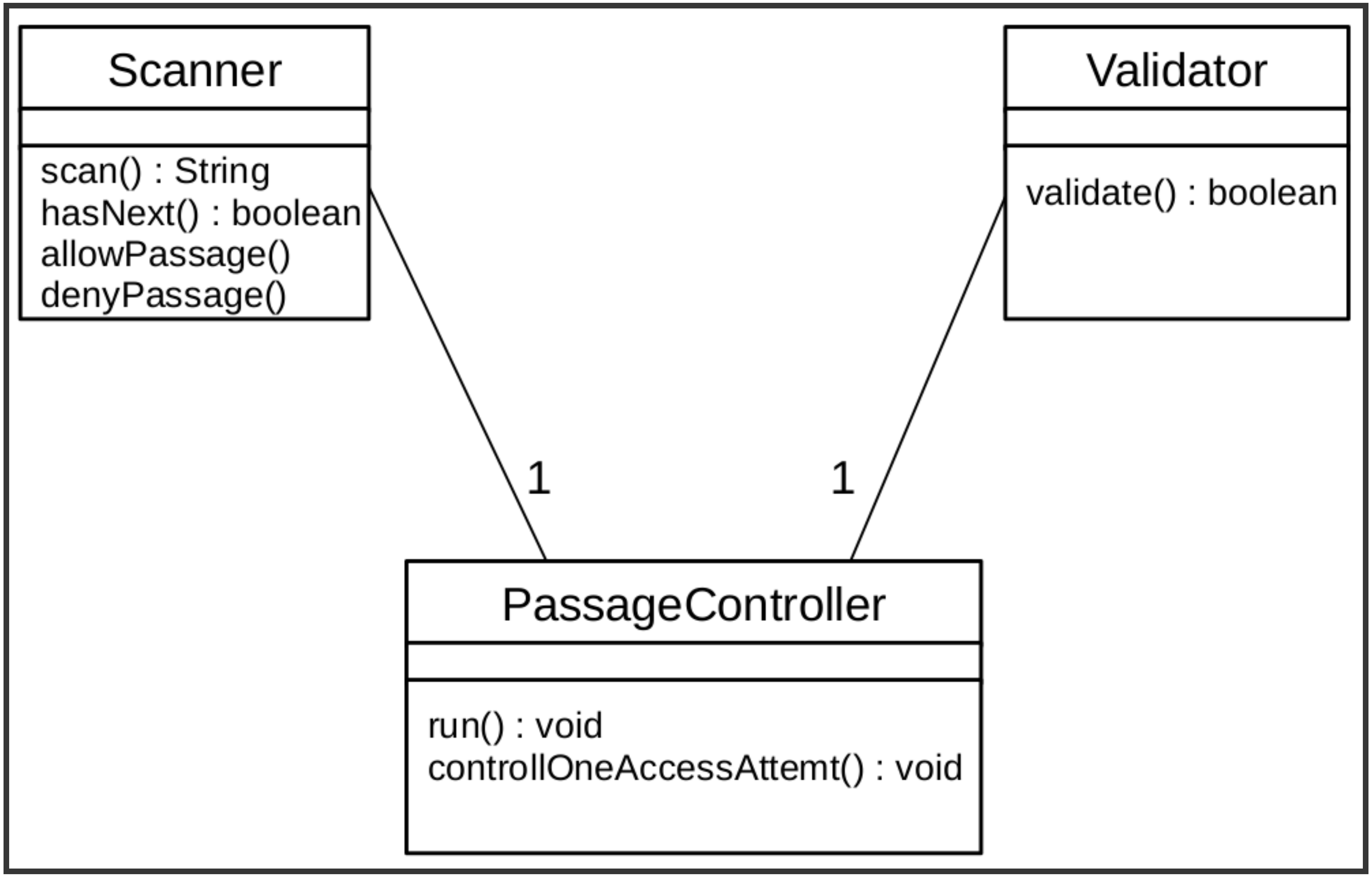
JUNIT

Presented by:
Pablo Gil, Xabier Moure, Marc Mocker and Samuel Navarro

Check out our examples

THE PROBLEM...


Scanner
scan() : String
hasNext() : boolean
allowPassage()
denyPassage()
Validator
validate() : boolean
1
1
PassageController
run() : void
controllOneAccessAttemt() : void

THE PROBLEM...

- our projects get bigger

THE PROBLEM...

- our projects get bigger
- the responsibilities are spread over various teams

THE PROBLEM...

- our projects get bigger
- the responsibilities are spread over various teams
- spaghetti code !!!

THE PROBLEM...

- our projects get bigger
- the responsibilities are spread over various teams
- spaghetti code !!!

who has to debug all the errors? and **how?**

WHAT ARE WE GOING TO DO?

TESTING

1. What is unit testing?
2. Structure of unit testing
3. simple tests - little demo
4. FIRST - *Requirements of unit testing*
5. `void` functions?
6. crazy tests - crazy demo

TESTING

WHAT IS TESTING?

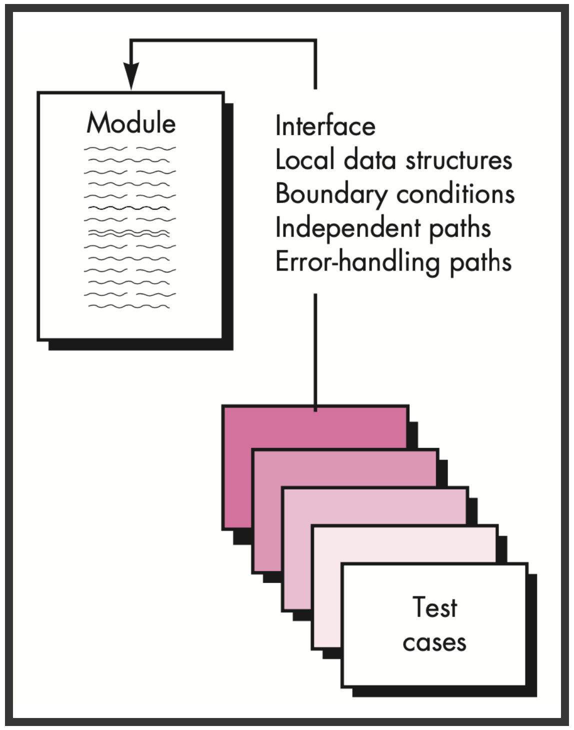
TESTING

Testing is a tool that developers have to uncover errors that are make inadvertently when software is designed and constructed.

Source: *Software Engineering. A Practitioners Approach (6th ed.). Roger Pressman*

UNIT TESTING

Verification of smallest unit of software

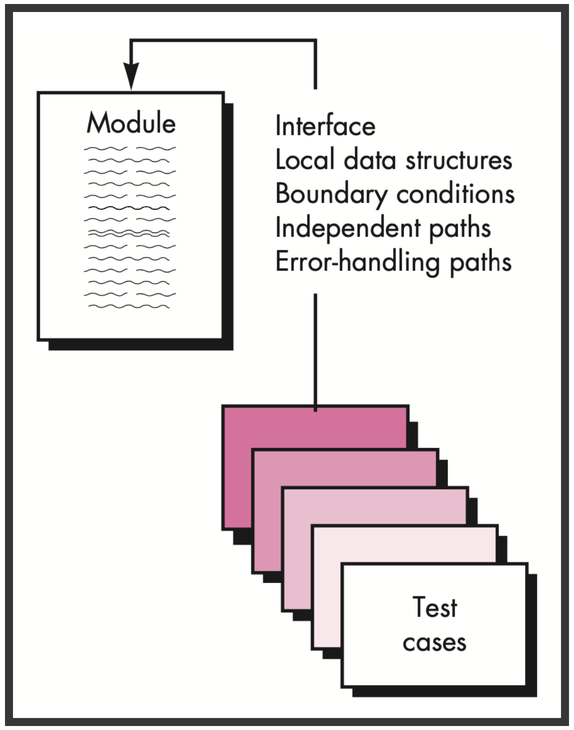


Source: *Software Engineering. A Practitioners Approach (6th ed.). Roger Pressman*

UNIT TESTING

Verification of smallest unit of software

In OOP

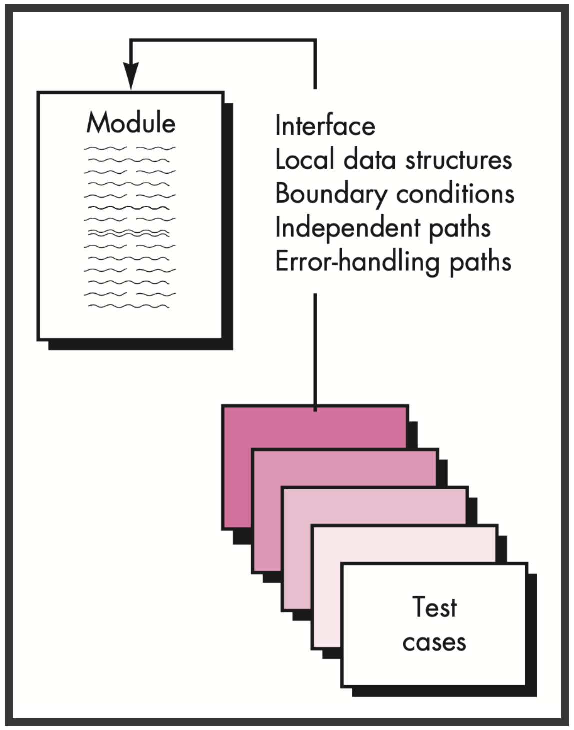


Source: *Software Engineering. A Practitioners Approach (6th ed.). Roger Pressman*

UNIT TESTING

Verification of smallest unit of software

Unit testing$^{\text{In OOP}}$

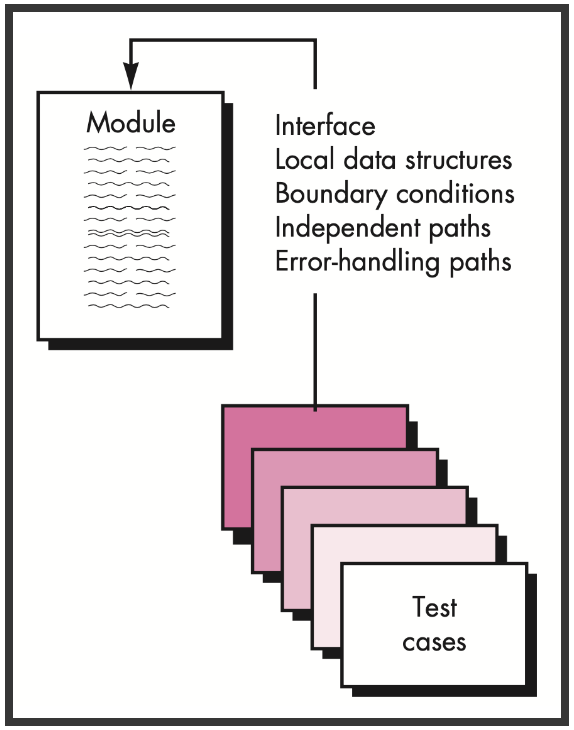


Source: *Software Engineering. A Practitioners Approach (6th ed.). Roger Pressman*

UNIT TESTING

Verification of smallest unit of software

Unit testing $\overset{\text{In OOP}}{\approx}$

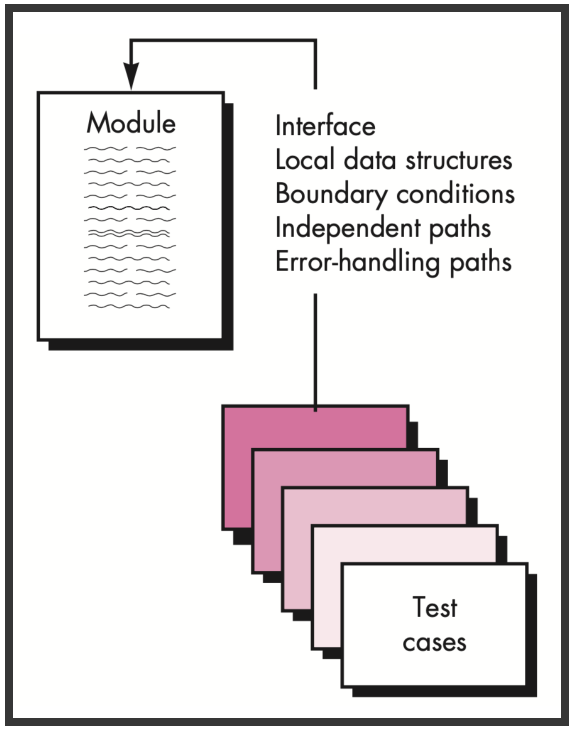


Source: *Software Engineering. A Practitioners Approach (6th ed.). Roger Pressman*

UNIT TESTING

Verification of smallest unit of software

Unit testing $\overset{\text{In OOP}}{\approx}$ Class testing



Source: *Software Engineering. A Practitioners Approach (6th ed.). Roger Pressman*

STRUCTURE OF UNIT TESTING

STRUCTURE OF UNIT TESTING

ARRANGE

STRUCTURE OF UNIT TESTING

ARRANGE

- Environment

STRUCTURE OF UNIT TESTING

ARRANGE

- Environment
- Parameters

STRUCTURE OF UNIT TESTING

ARRANGE

- Environment
- Parameters

STRUCTURE OF UNIT TESTING

ARRANGE

- Environment
- Parameters

ACT

STRUCTURE OF UNIT TESTING

ARRANGE

- Environment
- Parameters

ACT

- Executing

STRUCTURE OF UNIT TESTING

ARRANGE

- Environment
- Parameters

ACT

- Executing
- Saving results

STRUCTURE OF UNIT TESTING

ARRANGE

- Environment
- Parameters

ACT

- Executing
- Saving results

STRUCTURE OF UNIT TESTING

ARRANGE

- Environment
- Parameters

ACT

- Executing
- Saving results

ASSERT

STRUCTURE OF UNIT TESTING

ARRANGE

- Environment
- Parameters

ACT

- Executing
- Saving results

ASSERT

- Evaluation

STRUCTURE OF UNIT TESTING

ARRANGE

- Environment
- Parameters

ACT

- Executing
- Saving results

ASSERT

- Evaluation

```java
public class Pythagoras {
    public int triangle(int a, int b){
        if (a < 0 || b < 0){
            return -1;
        }
        return (int) Math.hypot(a, b);
    }
}
```

```java
public class Pythagoras {
    public int triangle(int a, int b){
        if (a < 0 || b < 0){
            return -1;
        }
        return (int) Math.hypot(a, b);
    }
}
```

```java
@Test
@DisplayName("simple execution with valid numbers")
void valid_numbers(){
    // ARRANGE
    Pythagoras p = new Pythagoras();
    int a = 3;
    int b = 5;

    // ACT
    var result = p.triangle(a, b);

    // ASSERT
    assertEquals(5, result);
}
```

```java
public class Pythagoras {
    public int triangle(int a, int b){
        if (a < 0 || b < 0){
            return -1;
        }
        return (int) Math.hypot(a, b);
    }
}
```

```java
@Test
@DisplayName("simple execution with valid numbers")
void valid_numbers(){
    // ARRANGE
    Pythagoras p = new Pythagoras();
    int a = 3;
    int b = 5;

    // ACT
    var result = p.triangle(a, b);

    // ASSERT
    assertEquals(5, result);
}
```

TEST PASSED

FIRST PRINCIPLE

FAST

I

R

S

T

- Perform speed of execution
- Easy to read
- No unnecessary additions

FIRST PRINCIPLE

F

ISOLATED

R

S

T

- One test result should not affect other tests
- Encapsulation (interface segregation principle)
- We are not testing our teammates code!

FIRST PRINCIPLE

F

I

REPEATABLE

- Deliver the same result on multiple executions

S

T

FIRST PRINCIPLE

F

I

R

SELF-VALIDATING

T

- Unequivocal result
- Developer should get a simple result with no need for interpretation
- There should not be the need to debug or test the tests!

FIRST PRINCIPLE

F

I

R

S

TIMELY

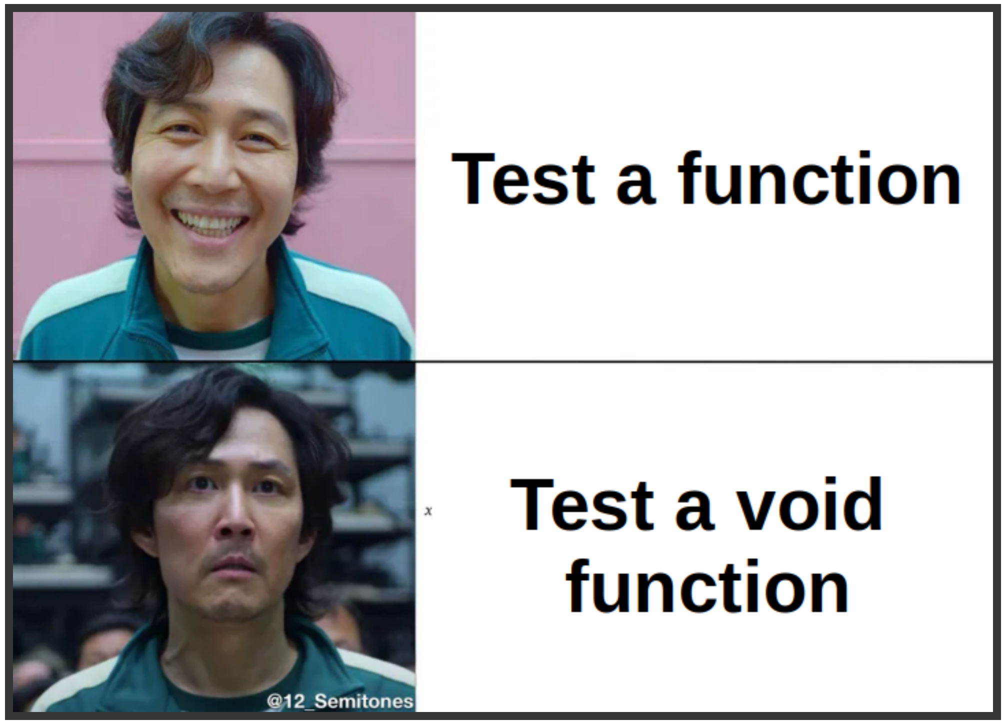
- Frequent testing (before each commit)
- Prevents the upstream repository from containing preventable bugs

NOW WE KNOW...

- the advantages of testing
- how to structure a test
- how to test a function which returns us any value

NOW WE KNOW...

- the advantages of testing
- how to structure a test
- how to test a function which returns us any value



`void` FUNCTIONS

We need to check if a `void` function has been executed properly.

`void` FUNCTIONS

We need to check if a `void` function has been executed properly.

STUBS

STUBS

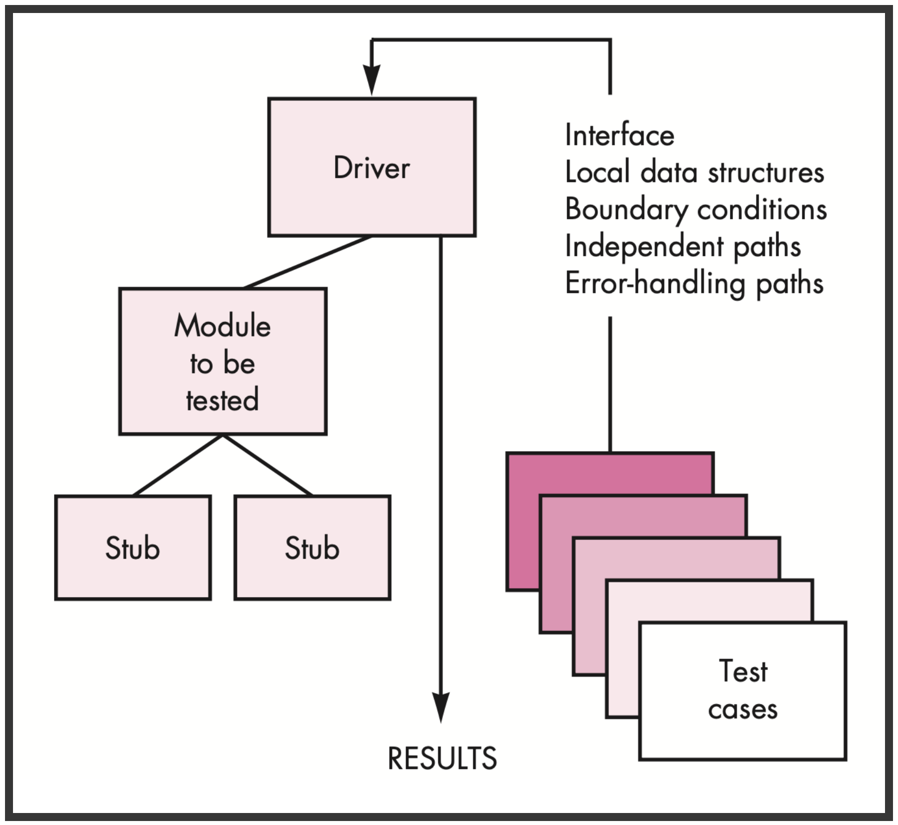
Object needed by the unit that we are testing.

It replaces that object emulating its functionality.

STUBS

Object needed by the unit that we are testing.

It replaces that object emulating its functionality.



Source: *Software Engineering. A Practitioners Approach (6th ed.). Roger Pressman*

REFERENCES

- **R. Pressman.** Software Engineering. A Practitioners Approach (6th ed.). *McGraw Hill*
- **I. Sommerville** Software Engineering (7th ed.). *Pearson*

Team Member	**Tasks**
Pablo Gil	Research on theory, realize slides, summarize theory
Xabier Moure	Research on theory, summarize theory, realize slides
Marc Mocker	Research on theory, prepare demos, realize slides
Samuel Navarro	Research on theory, realize slides, prepare demos

All group members have contributed with similar activities and effort on the different tasks